ROU: Advanced Keyword Search on Graph

Yifan Pan
School of Informatics and Computing
Indiana University
panyif@cs.indiana.edu

Yuqing Wu
School of Informatics and Computing
Indiana University
yuqwu@cs.indiana.edu

ABSTRACT

Keyword search, the major means for Internet search engines, has recently been explored in structured and semi-structured data. What is yet to be explored thoroughly is how optional and negative keywords can be expressed, what the results should be and how such search queries can be evaluated efficiently. In this paper, we formally define a new type of keyword search query, ROU-query, which takes as input keywords in three categories: *required*, *optional* and *unwanted*, and returns as output sets of nodes in the data graph whose neighborhood satisfies the keyword requirements. We define multiple semantics, including *maximal coverage* and *minimal footprint*, to ensure the meaningfulness of results. We propose query induced partite graph (QuIP), that can capture the constraints on neighborhood size and unwanted keywords, and propose a family of algorithms for evaluation of ROU-queries. We conducted extensive experimental evaluations to show our approaches are able to generate results for ROU-queries efficiently.

Categories and Subject Descriptors

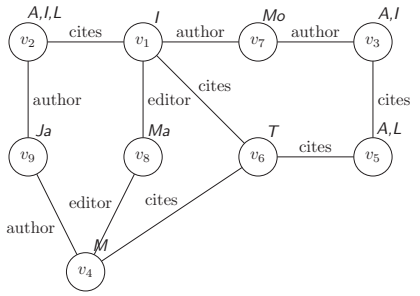
H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Query formulation*

Keywords

Keyword Search; Graph Data; Negative; Optional; Clique Enumeration

1. INTRODUCTION

Keyword search has been proven as an effective method for information retrieval, most notably used in search engines such as Google and Bing. While traditionally keyword search was focusing on finding particular entities (documents, images, videos, etc.) [8], recently we witnessed significant efforts on applying keyword search to structured and semi-structured data, including relational data [7], XML data [2] or general graphs [4], to find sets of data entries, in the form of Steiner trees [3, 4] and connected sub-graphs, such as r-radius Steiner graphs [6], keyword community [9] and r-cliques [5], that satisfy keyword and connectivity constraints. The common themes of most of these works are: (1) one set of keywords are given as input and the AND semantics is enforced (all keywords are required in each result); and (2) scoring functions based on node and edge weights are used to identify the top-k results.



Node	Content
v_1	***I*ntelligence**: Can Machines Beat Humans?
v_2	***A*rtificial *I*ntelligence** and Human ***L*anguage**
v_3	***A*rtificial *I*ntelligence** Handbook
v_4	Secrets of Human ***M*ind**
v_5	Natural ***L*anguage** & ***A*rtificial** Life
v_6	Machine ***T*ranslation**
v_7	Jouke ***Mo*lenaar**
v_8	Fabian ***Ma*rtinez**
v_9	Alexander ***Ja*cobsen**

Figure 1: Example Graph

[4] and [6], while still taking one keyword set as input, employed the OR semantics, allowing results to contain *some* keywords. But they cannot demand certain keywords *must* appear in every result. Other works on such semantics include [7] on relational databases and [2] on XML. However, they all assumed schema information, which is not applicable to general graphs. Meanwhile, negative predicate, while a key component in Boolean queries [8], has not been studied in the landscape of keyword search queries on graphs.

The issues listed above strictly limit what a traditional keyword search query can express. For example, consider the example graph shown in Figure 1 and the contents of the nodes shown in the table. All the content words can potentially be keywords for different queries; keywords used in our example queries throughout the paper are in **Bold** and the abbreviation marked in the figure. It is reasonable to ask questions such as:

Q.1 What correlated works by Molenaar and Jacobsen cover at least two topics from intelligence, language and mind?

Q.2 How is the concept of artificial intelligence presented in the context of language or mind, in current publications, without reference to articles about translation?

However, Existing works, even with ranking functions, cannot easily specify these queries.

CIKM'13, Oct. 27–Nov. 1, 2013, San Francisco, CA, USA.

http://dx.doi.org/10.1145/2505515.2505743.

Moreover, frequently users desire highly related results that bear *just the right amount of information*, which means:

RQ.1 the result does not include non-relevant information;

RQ.2 the result carries as rich information as possible;

RQ.3 the result does not include redundant information.

These requirements are not fully reflected by the ranking functions as proposed in [5–7, 9].

We take a different stand to address the open problems in both query specification and result definition as identified above. In this paper:

• We propose the ROU query on graph data, which allows users to specify three different keyword sets: *Required*, *Optional*, and *Unwanted* to integrate AND, OR and NOT semantics.

• We propose the *maximal coverage* and *minimal footprint* semantics for ROU queries, which are also well suited to be applied to other keyword search queries on graph data.

• We propose a novel data structure, called query induced partite graph, and two algorithms inspired by the Bron-Kerbosch algorithm [1] for efficient ROU query evaluation.

2. PROBLEM DEFINITION

The data graphs we study in this paper are node-labeled undirected graphs $G = (V_G, E_G, \lambda_G)$, where V_G is the set of nodes, $E_G \subseteq V_G \times V_G$ is the set of edges, and $\lambda_G : V_G \rightarrow 2^{\mathcal{L}}$ is a labeling function that maps each node in V_G to a set of keyword terms in $\mathcal{L}$. A node v *contains* a keyword k if $k \in \lambda_G(v)$. We overload the mapping function $\lambda_G()$ so when it is applied to a set of nodes $V \subseteq V_G$, it returns the union of the keyword sets of the nodes in V, i.e. $\lambda_G(V) = \bigcup_{v \in V} \lambda_G(v)$.

2.1 Keyword-based Constraints

Given a set of nodes V in G, we measure the size of V by the distance among the nodes in G.

DEFINITION 2.1. **[Bypass Distance & Diameter]** *Given a keyword set K, for any nodes $u, v \in V_G$, the distance between u and v, $dis(u,v)$, is the length of the shortest path between u and v in G; the K-bypass distance between u and v, $\overline{dis}^K(u,v)$, is the length of the shortest path between u and v that does not pass through any node that contains a keyword in K. The K-bypass diameter of $V \subseteq V_G$ is*

$$\overline{d}^K(V) = \max_{u,v \in V} \overline{dis}^K(u,v)$$

Please note that $\overline{dis}^K(u,v) = \infty$ if at least one keyword in K appears on each path between u and v. Also observe that $\overline{dis}^{\emptyset}(u,v) = dis(u,v)$.

Consider a set of nodes that are close to each other and do not involve the unwanted keyword terms. We now consider how to identify whether the set provides *just the right amount of information*, given positive keyword constraints.

DEFINITION 2.2. **[Cover & h-cover]** *Given a set of nodes $V \subseteq V_G$, a set of keyword terms K ($K \neq \emptyset$), and a threshold h, $0 \leq h \leq 1$, we say V h-cover K (denoted $V \succ^h K$), if $\frac{|K \cap \lambda_G(V)|}{|K|} \geq h$ and for every $v \in V$, $\lambda_G(v) \cap K \neq \emptyset$; when $V \succ^1 K$ stands, we say V cover K (denoted $V \succ K$).*

EXAMPLE 2.1. *Consider the sample data graph in Figure 1. $dis(v_1, v_5) = 2$; $\overline{dis}^{\{T\}}(v_1, v_5) = 3$. Consider a keyword set K=$\{A, I, L, M\}$, then, $\{v_3, v_4, v_5\} \succ K$, $\{v_3, v_5\} \not\succ K$, and $\{v_1, v_5\} \succ^{\frac{3}{4}} K$, $\{v_1, v_3\} \not\succ^{\frac{3}{4}} K$.*

The "*for every $v \in V$, $\lambda_G(v) \cap K \neq \emptyset$*" clause in Def. 2.2 ensures the satisfaction of RQ.1. RQ.3 is fulfilled automatically when $|\lambda_G(v)| = 1$ holds for all nodes in G. However, when the label of a node contains multiple keyword terms, the situation becomes much more delicate. Existing works [5, 9], under such circumstances, chose to take a vague stand on how each node in the result represents the keywords. Here, we will define the search results of the ROU queries in a more precise manner.

2.2 ROU Keyword Search Query

In the ROU query we introduce, users can specify keyword constraints in three different categories: the *Required* set of keywords which they want to *all appear* in each result; the *Optional* set of keywords which they want to *at least partially appear* in each result; and the *Unwanted* set of keywords which *should not be associated with* the results. We use K_r, K_o and K_u to represent them, respectively. In addition, associated with the optional keyword set is a threshold h, a real number between 0 and 1. And a constraint dis_{MAX} needs to be specified to regulate the size of each result measured in terms of node set diameter.

DEFINITION 2.3. **[ROU Keyword Search Query]** *Given a data graph $G = (V_G, E_G, \lambda_G)$, an ROU query is specified in the form of $q = (K_r, K_o, K_u, h, dis_{MAX})$, in which $K_r \cup K_o \neq \emptyset$; $0 \leq h \leq 1$ when $K_o \neq \emptyset$; and $dis_{MAX} \geq 0$.*

The result of evaluating q on G is a set of node sets, i.e. $q(G) \in 2^{V_G}$. Each $V \in q(G)$ must satisfy:

1. *if $K_r \neq \emptyset$, there exists $V_r \subseteq V$, such that $V_r \succ K_r$;*
2. *if $K_o \neq \emptyset$, there exists $V_o \subseteq V$, such that $V_o \succ^h K_o$;*
3. *$V = V_r \cup V_o$;*
4. *$\overline{d}^{K_u}(V) \leq dis_{MAX}$.*

EXAMPLE 2.2. *Consider Q.1-2 presented in Introduction, they can be specified as ROU queries:*

Q.1: $q_1 = (\{Mo, Ja\}, \{I, L, M\}, \emptyset, \frac{2}{3}, 3)$

Q.2: $q_2 = (\{A, I\}, \{L, M\}, \{T\}, \frac{1}{2}, 2)$

Applying q_2 to the data graph shown in Figure 1, we have

$$q_2(G) = \{\{v_1, v_2, v_4\}, \{v_1, v_2\}, \{v_2, v_4\}, \{v_3, v_5\}, \{v_2\}\}.$$

We would like to bring readers' attention to two types of subset relationships among node sets in $q_2(G)$:

Case 1. $\{v_1, v_2\} \subset \{v_1, v_2, v_4\}$, $\{v_1, v_2\} \succ \{A, I, L\}$, and $\{v_1, v_2, v_4\} \succ \{A, I, L, M\}$.

Case 2. $\{v_2, v_4\} \subset \{v_1, v_2, v_4\}$, $\{v_1, v_2, v_4\} \succ \{A, I, L, M\}$, and $\{v_2, v_4\} \succ \{A, I, L, M\}$.

As can be seen from the example above, some resultant node sets are consumed by others: (1) a node set is by itself not rich enough, as $\{v_1, v_2\}$ in Case 1, violating RQ. 2; or (2) a node set contains redundant information, as $\{v_1, v_2, v_4\}$ in Case 2, violating RQ. 3.

Intuitively, when a set of keywords are partially covered (h-cover), the result that covers more keywords is considered to carry richer information comparing to the one that covers less, hence should be favored over the latter.

DEFINITION 2.4. **[Coverage Comparison]** *Given a set of keywords K, consider two node sets $V, V' \subseteq V_G$. We say that V's coverage of K is consumed by V''s coverage of K, denoted $V <_{cv}^K V'$, if $V \subset V'$ and $\lambda_G(V) \cap K \subset \lambda_G(V') \cap K$.*

DEFINITION 2.5. **[Maximal Coverage Semantics]** *Given a data graph $G = (V_G, E_G, \lambda_G)$, an ROU query $q = (K_r, K_o, K_u, h, dis_{MAX})$, the maximal coverage of the query result $q(G)$ is a subset of $q(G)$, defined as*

$$\widehat{q(G)} = q(G) - \{V | \exists V' \in q(G)(V <_{cv}^{K_r \cup K_o} V')\}.$$

The maximal coverage semantics introduced above does not guarantee that each resultant node set is the minimal needed to cover the positive keywords ($K_r \cup K_o$) in query q. As a remedy, we introduce the *minimal footprint* semantics, which ensures smallest resultant node set by favoring a smaller node set over a bigger one, when the two sets cover exactly the same sets of positive keyword terms. Formally,

DEFINITION 2.6. **[Footprint Comparison]** *Given a set of keywords K, consider two node sets $V, V' \subseteq V_G$ which cover the same subset of K, i.e. $\lambda_G(V') \cap K = \lambda_G(V) \cap K$. If $V \subset V'$, we say that V has a smaller footprint than V' in covering K, denoted $V <_{fp}^{K} V'$.*

DEFINITION 2.7. **[Minimal Footprint Semantics]** *Given a data graph $G = (V_G, E_G, \lambda_G)$, an ROU query $q = (K_r, K_o, K_u, h, dis_{MAX})$, the minimal footprint of the query result $q(G)$ is a subset of $q(G)$, defined as*

$$\underbrace{q(G)} = q(G) - \{V | \exists V' \in q(G)(V' <_{fp}^{K_r \cup K_o} V)\}.$$

Taking the concepts defined above all into consideration, we define the *condensed* semantics of ROU query, which satisfy RQ. 1, RQ. 2 and RQ. 3, as follows:

DEFINITION 2.8. **[Condensed Semantics]** *Given a data graph G, an ROU query $q = (K_r, K_o, K_u, h, dis_{MAX})$, the condensed semantics computes $\widehat{q(G)} = \widehat{q(G)} \cap \underbrace{q(G)}$.*

EXAMPLE 2.3. *Consider again query q_2 on G, then*

$$\begin{aligned} \widehat{q_2(G)} &= \{\{v_1, v_2, v_4\}, \{v_2, v_4\}, \{v_3, v_5\}\} \\ \underbrace{q_2(G)} &= \{\{v_2, v_4\}, \{v_3, v_5\}, \{v_2\}\} \\ \widehat{q_2(G)} &= \{\{v_2, v_4\}, \{v_3, v_5\}\} \end{aligned}$$

Note that a node set does not have to cover all the keywords in $K_r \bigcup K_o$ to qualify for $\widehat{q(G)}$. $\{v_3, v_5\}$ is one such example.

3. QUERY INDUCED PARTITE GRAPH

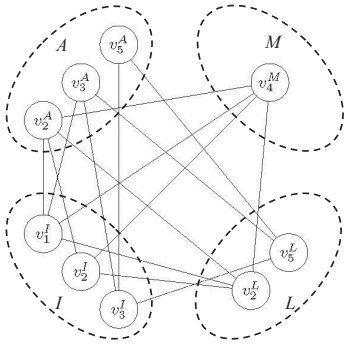


Figure 2: QuIP of q_2 on G

We propose Query Induced Partite Graph (QuIP) as an intermediate data structure for efficient answering of ROU queries. One key notion introduced in QuIP is *shadowing*. Given a data graph and a query q, for each node that covers a positive keyword in q, we create a copy of it for each positive keyword it covers, and label the copy with the keyword. These copies are the nodes in QuIP. (We refer to the nodes and edges in QuIP as t-nodes and t-edges.) Two t-nodes are adjacent in QuIP if their labels are different and (1) they are copies of the same original data node; or (2) the K_u-bypass distance between the two original data nodes they represent are within the size constraint (i.e. dis_{MAX}) in G. Formally,

DEFINITION 3.1. **[Query Induced Partite Graph]** *Given a data graph $G = (V_G, E_G, \lambda_G)$ and an ROU query $q = (K_r, K_o, K_u, h, dis_{MAX})$, the* Query Induced Partite Graph *of q on G, denoted $G_T(q, G) = (V_T, E_T, \lambda_T)$ is constructed as follows:*

- $V_T = \{v^k | v \in V_G \wedge k \in (K_r \cup K_o) \cap \lambda_G(v)\}$;
- $\lambda_T(v^k) = k$, *for all* $v^k \in V_T$;
- $(v^k, u^l) \in E_T$ *if* $k \neq l$ *and (1)* $v = u$; *or (2)* $v \neq u \wedge \overline{dis}^{K_u}(v, u) \leq dis_{MAX}$.

Given a node v in data graph G, we call all t-nodes $v^k \in V_T$ the *shadows* of v. And for each such $v^k \in V_T$, we say that v is its *base*. We define the *base*() and *shadow*() functions to represent the mapping:

$$\begin{aligned} shadow(v) &= \{v^k | k \in (K_r \cup K_o) \cap \lambda_G(v)\} \\ base(v^k) &= v \\ base(V'_T) &= \bigcup_{v^k \in V'_T} base(v^k) \\ base(G'_T) &= base(V'_T) \text{ where } G'_T \text{ is a subgraph of } G_T \\ base(S) &= \{base(G'_T) | G'_T \in S\} \\ & \quad \text{where S is a set of subgraphs of } G_T \end{aligned}$$

In a QuIP, given a keyword k, we call the set of all t-nodes labeled k the k-cluster, denoted $V_T^k = \{v^k | v^k \in V_T\}$.

EXAMPLE 3.1. *Again consider query q_2 and sample graph G. $G_T(q_2, G)$ is shown in Figure 2. There are two shadow nodes of v_3: ${v_3}^A$ and ${v_3}^I$, i.e. $shadow(v_3) = \{{v_3}^A, {v_3}^I\}$. There are four k-clusters, shown in dotted circles.*

The following properties of query induced partite graph can be established naturally from its definition.

OBSERVATION 3.1. *Given a QuIP $G_T(q, G) = (V_T, E_T, \lambda_T)$ of ROU query q on G, the following statements hold:*

- $V_T = \bigcup_{k \in K_r \cup K_o} V_T^k$;
- $V_T^k \cap V_T^{k'} = \emptyset$ *if* $k \neq k'$;
- *for any* $u^k, v^k \in V_T^k$, $(u^k, v^k) \notin E_T$.
- *given a t-node set $V'_T \subseteq V_T$, if the induced graph of V'_T is a clique, then,*
  - *for any two t-nodes $v^k, u^l \in V'_T$, v^k, u^l cannot belong to the same keyword cluster, i.e. $k \neq l$;*
  - $|V'_T| \leq |K_r \cup K_o|$;

We now show how to take advantage of the information represented in a QuIP to efficiently answer an ROU query.

EXAMPLE 3.2. *From Example 2.3, we know $\widehat{q_2(G)} = \{\{v_2, v_4\}, \{v_3, v_5\}\}$. In $G_T(q_2, G,)$, we find that the two node sets are both bases of maximal cliques, with $base(\{{v_5}^A, {v_3}^I, {v_5}^L\}) = \{v_3, v_5\}$ and $base(\{{v_2}^A, {v_2}^I, {v_2}^L, {v_4}^M\}) = \{v_2, v_4\}$.*

Note that different sets of t-nodes can be mapped to the same base. For example, consider $V'_T = \{{v_5}^A, {v_3}^I, {v_5}^L\}$ and $V''_T = \{{v_3}^A, {v_3}^I, {v_5}^L\}$, $base(V'_T) = base(V''_T) = \{v_3, v_5\}$.

However, not all the maximal cliques in $G_T(q_2, G)$ correspond to members of $\widehat{q_2(G)}$. For instance, $\{{v_3}^A, {v_1}^I\}$ is

a maximal clique, but it does not contain enough keywords so its base $\{v_1, v_3\}$ does not even belong to $q_2(G)$. Another example is $\{{v_2}^A, {v_1}^I, {v_2}^L, {v_4}^M\}$. Its base is $\{v_1, v_2, v_4\}$ and satisfies maximal coverage semantics, but it does not satisfy the minimal footprint semantics as it covers no more keywords than $\{v_2, v_4\}$ does.

OBSERVATION 3.2. *Given $G_T(q, G) = (V_T, E_T, \lambda_T)$:*

- *If node set $V \in \widehat{q(G)}$, then, there must exist a maximal clique c in $G_T(q, G)$, such that $base(c) = V$.*
- *There may exist maximal clique c in $G_T(q, G)$, $base(c) \notin q(G)$.*
- *There may exist maximal cliques c, c' in $G_T(q, G)$, such that $base(c) <^K_{fp} base(c')$, where $K = \lambda_T(base(c)) = \lambda_T(base(c'))$.*

We can see that finding all maximal cliques in QuIP is not enough for answering an ROU query. We introduce the *h-cover clique* to address these issues.

DEFINITION 3.2. [**h-cover clique**] *Given $G_T(q, G)$, $G'_T = (V'_T, E'_T, \lambda_T)$ is a subgraph of G_T induced by t-node set V'_T. We say that G'_T is a h-cover clique of q if*

- *G'_T is a clique;*
- *there exists $V'_{rT} \subseteq V'_T$ and $V'_{rT} \succ K_r$, if $K_r \neq \emptyset$;*
- *there exists $V'_{oT} \subseteq V'_T$ and $V'_{oT} \succ^h K_o$, if $K_o \neq \emptyset$; and*
- *$V'_T = V'_{rT} \cup V'_{oT}$.*

Given a QuIP $G_T(q, G)$, we use $hcClq(G_T)$ [1] to represent the set of all h-cover cliques in G_T.

THEOREM 3.1. *Given $G_T(q, G)$, which is the QuIP of q on G, the following holds:*

$$\widehat{q(G)} \subseteq base(hcClq(G_T)) \subseteq q(G).$$

Theorem 3.1 establishes QuIP as a suitable vehicle for answering ROU queries. We are interested in following subsets of $hcClq(G_T)$:

$$\begin{aligned}\overbrace{hcClq(G_T)} \quad &= hcClq(G_T) \\ &-\{c | c \in hcClq(G_T) \wedge \exists c' \in hcClq(G_T) \\ &\quad (V_c \subset V'_c \wedge \lambda_T(V_c) \cap K \subset \lambda_T(V'_c) \cap K)\}\end{aligned}$$

is a subset of $hcClq(G_T)$ that guarantees maximal coverage.

$$\begin{aligned}\widehat{hcClq(G_T)} \quad &= \overbrace{hcClq(G_T)} \\ &-\{mc | mc \in \overbrace{hcClq(G_T)} \wedge \exists c \in hcClq(G_T) \\ &\quad (base(c) \subset base(mc) \wedge \lambda_T(c) \cap K = \lambda_T(mc) \cap K)\}\end{aligned}$$

is a subset of $\overbrace{hcClq(G_T)}$ that guarantees minimal footprint.

COROLLARY 3.1. *Given $G_T(q, G)$, the following holds:*

$$\begin{aligned}base(\overbrace{hcClq(G_T(q, G))}) &= \overbrace{q(G)} \\ base(\widehat{hcClq(G_T}(q, G))) &= \widehat{q(G)}.\end{aligned}$$

EXAMPLE 3.3. *Consider query q_2 and four sets of t-nodes: $V_{T_1} = \{{v_3}^A, {v_1}^I\}$, $V_{T_2} = \{{v_2}^A, {v_1}^I, {v_2}^L\}$, $V_{T_3} = \{{v_2}^A, {v_1}^I, {v_2}^L, {v_4}^M\}$, and $V_{T_4} = \{{v_2}^A, {v_2}^I, {v_2}^L, {v_4}^M\}$, each inducing a clique in $G_T(q_2, G)$, we call them $c_1 \ldots c_4$, respectively.*

$c_1 \notin hcClq(G_T(q_2, G))$ as V_{T_1} does not cover adequate keywords, and $base(V'_T) = \{v_1, v_3\} \notin q(G)$.

$c_2 \in hcClq(G_T(q_2, G))$, however $c_2 \notin \overbrace{hcClq(G_T(q_2, G))}$ as c_2 is a sub-graph of c_3, which covers more keywords.

$c_3 \in \overbrace{hcClq(G_T(q_2, G))}$, however, $c_3 \notin hcClq(\widehat{G_T}(q_2, G))$, as c_3 and c_4 covers the same set of keywords, and $base(c_4) \subset base(c_3)$.

$c_4 \in hcClq(\widehat{G_T}(q_2, G))$, and $base(c_4) \in \widehat{q(G)}$.

Based on Theorem 3.1 and Corollary 3.1, we can use QuIP to answer an ROU query in two steps: (1) constructing QuIP based on an incoming query; and (2) identifying $\widehat{hcClq(G_T)}$ whose base is member of $\widehat{q(G)}$.

Neighborhood index is often used to answer whether two nodes are connected within a distance threshold [5]. The technique can be extended to include labels to deduce whether two nodes are connected, bypassing K_u. Such extension works for small K_u, but is not practical when $|K_u|$ is large. Designing better data structures and algorithms for QuIP construction is left for future work, and we consider cases where $|K_u| \leq 1$ in our experiments. Here we focus on step (2), using QuIP to compute $\widehat{q(G)}$.

4. $\widehat{q(G)}$ GENERATION

4.1 Base Algorithm

Enumerating maximal cliques in a graph is a well studied problem in discrete mathematics [1, 11]. We adapt the Bron-Kerbosch algorithm [1] to find maximal cliques in QuIP. The details of the recursive algorithm is shown in Algo. 2. The process of generating the full set of $\widehat{q(G)}$ starts by triggering the `cdsCliqueEnum()` function, with P initialized to be V_T, and R and X empty set, is shown in Algo. 1.

Algorithm 1: $\widehat{q(G)}$ Generation

Data: $G_T(q, G)$
Result: $\widehat{q(G)}$
1 $R \leftarrow \emptyset$; $P \leftarrow V_T$; $X \leftarrow \emptyset$;
2 `cdsCliqueEnum` (R, P, X) ;

`cdsCliqueEnum()`, whose details are presented in Algo. 2, is amended with two pruning functions:

• `satisfyQ()` handles positive keyword constraints. Given a set of t-nodes $V \subseteq V_T$, `satisfyQ(V)=TRUE` if for some $V_r \subseteq V$, $V_r \succ K_r$ (if $K_r \neq \emptyset$) and for some $V_o \subseteq V$, $V_o \succ^h K_o$ (if $K_o \neq \emptyset$). It guarantees that all the maximal cliques enumerated by the algorithms are h-cover cliques.

• `notMnfp()` checks if there exists a node $v \in base(R)$ such that $\lambda_G(base(R)) \cap (K_r \cup K_o) = \lambda_G((base(R) - \{v\})) \cap (K_r \cup K_o)$. If so, the base of R cannot possibly be a subset of any results that satisfy the minimal footprint requirement as we can always remove v for a more "concise" result.

In Algo. 2, P is a set holding potential t-nodes that can be added to the current partial clique R. In the recursion process, t-nodes in P will always be neighbors to all the t-nodes in R. Thus whenever P is not empty, we can always pick a t-node from it to add to R to make a bigger clique. A t-node v is added to X only after all the maximal cliques containing $R \cup \{v\}$ are explored (line 12). Thus when P is empty and X is not, it implies that a clique that contains t-nodes in $R \cup X$ has already been considered before. So a clique that contains only t-nodes in R is not maximal, and

[1] We abbreviate $G_T(q, G)$ as G_T when there is no ambiguity about the parameters q and G.

Level	R	P	X	R'	P'	X'	Action
0	$\emptyset$	V_T	$\emptyset$	$\{{v_3}^I\}$	$\{{v_3}^A, {v_5}^A, {v_5}^L\}$	$\emptyset$	recursive call
1	$\{{v_3}^I\}$	$\{{v_3}^A, {v_5}^A, {v_5}^L\}$	$\emptyset$	$\{{v_3}^I, {v_5}^A\}$	$\{{v_5}^L\}$	$\emptyset$	recursive call
2	$\{{v_3}^I, {v_5}^A\}$	$\{{v_5}^L\}$	$\emptyset$	$\{{v_3}^I, {v_5}^A, {v_5}^L\}$	$\emptyset$	$\emptyset$	recursive call
3	$\{{v_3}^I, {v_5}^A, {v_5}^L\}$	$\emptyset$	$\emptyset$				report result, backtrack
2	$\{{v_3}^I, {v_5}^A\}$	$\emptyset$	$\{{v_5}^L\}$				backtrack
1	$\{{v_3}^I\}$	$\{{v_3}^A, {v_5}^L\}$	$\{{v_5}^A\}$	$\{{v_3}^I, {v_3}^A\}$	$\{{v_5}^L\}$	$\emptyset$	recursive call
2	$\{{v_3}^I, {v_3}^A\}$	$\{{v_5}^L\}$	$\emptyset$	...	...	...	...

Table 1: Evaluating q_2 Using Algo 2

our algorithm needs to backtrack. When both P and X are empty, a maximal clique in $G_T(q, G)$ has been formed; and it belongs to $hc\widehat{Clq(G_T)}$ because of `notMnfp()`. So we can report its base, which belongs to $\widehat{q(G)}$, as a query result. An execution example of the algorithm can be found in Table 1.

Algorithm 2: cdsCliqueEnum(R,P,X)

Data: R, P, X
Result: bases of cliques that contain all t-nodes in R, some t-nodes in P and no t-nodes from X, satisfy condensed semantics

```
1  if notMnfp (R) then
2  |  return
3  end
4  if P is empty and X is empty then
5  |  output base(R) as a result; return
6  end
7  for each t-node v in P do
8  |  R' ← R ∪ {v}; P' ← P ∩ N(v); X' ← X ∩ N(v);
9  |  if satisfyQ (R' ∪ P') then
10 |  |  cdsCliqueEnum (R', P', X')
11 |  end
12 |  P ← P − {v} ; X ← X ∪ {v} ;
13 end
```

4.2 Optimization

Though `cdsCliqueEnum` generates exactly $\widehat{q(G)}$, it may consider cliques that consist of different t-nodes but share the same base, generating duplicated results. For example, applying the algorithm to $G_T(q_2, G)$, it will generate result $\{v_3, v_5\}$ twice, one from clique $\{{v_5}^A, {v_3}^I, {v_5}^L\}$, another from $\{{v_3}^A, {v_3}^I, {v_5}^L\}$.

Algorithm 3: cdsCliqueEnumGreedy(R,P,X)

```
   // Lines 1-6 same as Lines 1-6 in cdsCliqueEnum
7  for each t-node v in P do
8  |  R' ← R; P' ← P; X' ← X;
9  |  for each t-node v' in P∩ shadow(base(v)) do
10 |  |  R' ← R' ∪ {v'}; P' ← P' ∩ N(v');
   |  |  X' ← X' ∩ N(v');
11 |  end
12 |  if satisfyQ (R' ∪ P') then
13 |  |  cdsCliqueEnum (R', P', X')
14 |  end
15 |  for each t-node v' in P∩ shadow(base(v)) do
16 |  |  P ← P − {v'} ; X ← X ∪ {v'} ;
17 |  end
18 end
```

To address this issue, we introduce `cdsCliqueEnumGreedy`, presented in Algo. 3, which can generate $\widehat{q(G)}$ without considering duplicated cliques. The basic idea is that when a t-node v is picked from P and put into R, we also put all shadows of the base of v that are in P, i.e. $P \cap shadow(base(v))$, into R (lines 9-11), and adjust P and X accordingly. When the search is done on the current level, all these t-nodes will be moved to X to prevent cliques with same base from being considered in the future (lines 15-17). Thus if we apply `cdsCliqueEnumGreedy`, assuming at the beginning in Table 1 that we pick ${v_3}^A$ from P at level 0, then both ${v_3}^A$ and ${v_3}^I$ will be added to R. When the execution terminates, only clique induced by $\{{v_3}^A, {v_3}^I, {v_5}^L\}$ will be generated, and $\{{v_5}^A, {v_3}^I, {v_5}^L\}$ will be skipped.

4.3 Discussion

Theoretically the complexity of enumerating maximal cliques in a graph with n nodes is $O(3^{\frac{n}{3}})$, as stated in [10]. However, the complexity of our algorithms is much lower. In the worst case, the complexity is $O((\max_{k \in K_r \bigcup K_o} |V_T^k|)^{|K_r \bigcup K_o|})$, where $|V_T^k| \ll |V_G|$ and $|K_r \bigcup K_o|$ is small. Moreover, due to the partite nature and sparsity of $G_T(q, G)$, it is often the case that in $G_T(q, G)$ each node is adjacent to only a few nodes. Under such circumstance, the recursion tree is shallow and subtrees are small below the first level of recursion. Hence, the average complexity is very close to $O(|V_T|)$.

5. EXPERIMENTAL EVALUATIONS

We conduct our experimental evaluation on the Proximity DBLP database, which is based on data from the DBLPs. The data contain 1.1M nodes and 1.8M edges, and the total number of node pairs (u, v) that satisfies $dis(u, v) \leq 3$ is 111M. Based on term closeness and keyword frequency, we organize keywords in groups to form queries that are representative enough to better interpret the impact of various of input parameters. We will present results of ROU queries whose keywords are picked from those in the table below.

	Keywords
1	bayesian, inference, statistical, polynomial, propagation, belief, graphical
2	arti cial, intelligence, reasoning, vision, learning, robotics
3	compiler, type, programming, semantics, grammar
4	database, query, optimization, sql, datalog
5	operating, system, parallel, distributed, memory

Our algorithms are written in Java and our experiments are run on a laptop running Windows 7 with Intel®Core^{TM} i5-2450M CPU @ 2.50GHz and 8GB memory, in which a maximum of 4GB memory are dedicated to the JVM.

QuIP Size. The QuIP sizes of some example queries are shown in Figure 3 (with dis_{MAX}= 3). In general, $QuIP$ is significantly smaller than the data graph, both in terms of node number and edge number. In addition, the presence of

Query	$K_r \cup K_o$		K_u		Nodes	Edges
	size	groups	size	groups		
Q_1	6	G1	0		13743	61538
Q'_1	6	G1	1	G1	13498	58208
Q_2	5	G2	0		13576	35713
Q'_2	5	G2	1	G2	13464	35362
Q_3	5	G3	0		12037	132K
Q_4	5	G4	0		13764	2.5M
Q'_4	5	G4	1	G4	12853	1.8M
Q_5	5	G5	0		56976	1.2M
Q_6	5	G1-5	0		8262	79K

Figure 3: QuIP Size

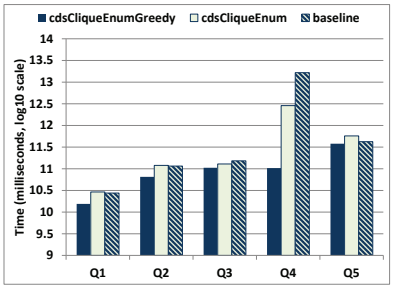


Figure 4: Algo 2. vs Algo. 3

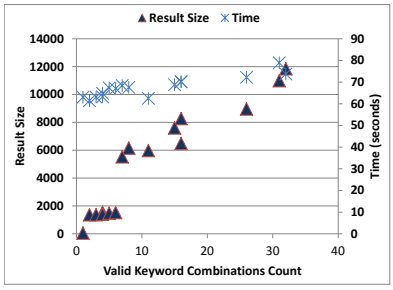


Figure 5: Positive Keywords Impact

negative keyword constraint, even when the size of the keyword set is 1, can reduce the size of $QuIP$ (and subsequently lead to decrease of enumeration time). In the following, we are presenting results on a variety of graphs that are of different size, i.e. graphs generated by Q_1 to Q_5.

Algorithm Performance Comparisons. We compare the performance of algorithm `cdsCliqueEnum` and algorithm `cdsCliqueEnumGreedy`, with the baseline algorithm, which uses the Bron-Kerbosch algorithm to generate all maximal cliques on the QuIP, then verifies the keyword constraints and the condensed semantics. The performance comparison of a selected set of queries is shown in Figure 4. Please note that log-scale is used for y-axis. For Query 3-5, we cut off the computation at 100K results as the potential result size can be huge.

As can be seen, the `cdsCliqueEnumGreedy` algorithm generally outperforms the baseline algorithm, cutting execution time by half, while the `cdsCliqueEnum` algorithm has comparable performance as the baseline. In cases, such as Q_4, where very few among large number of maximal cliques in the QuIP satisfy the keyword constraints and condensed semantics, the pruning in both our algorithms are effective, and `cdsCliqueEnumGreedy` particularly outperforms the baseline algorithm by two orders of magnitude. Theoretically, `cdsCliqueEnum` algorithm may outperform `cdsCliqueEnumGreedy` when the overhead of the additional pruning employed by the latter overshadow the benefit of such pruning, however, such cases are extremely rare according to our experimental study.

Impact of Positive Keyword Constraints. To measure the impact of the distribution of the keywords among K_r and K_o on result size and query performance, we introduce the notion of *valid keyword combinations count*, which is a function of K_o and h, i.e. $Ct(K_o, h) = \sum_{n=\lceil |K_o| \times h \rceil \ldots |K_o|} C^n_{|K_o|}$. We randomly pick sets of keyword terms, and for each set, vary the distribution of these keywords among K_r and K_o, and the h value, then, compare the result sizes and query evaluation time among the queries that share the same $K_r \cup K_o$. Figure 5 shows the results on a set of 6 keywords picked from group 2.

Please note that when the valid keyword combinations count increases, the evaluation time increases at a much slower pace than the result size. This is due to the fact that the cliques exploited in the evaluation process share common sub-clique, and the recursive nature of our algorithms are able to absorb the extra computational cost.

6. CONCLUSION AND FUTURE WORK

In this paper we initialized a research topic by introducing ROU, a new type of keyword search queries on graph data, which allows user to specify both positive and negative keyword constraints. We formally defined the semantics in terms of maximal coverage and minimal footprint. We proposed *Query Induced Partite Graph* (QuIP) for representing candidate data entries and their relationships, and algorithms that take advantage of information collected in QuIP to efficiently answer ROU queries efficiently. However, the problem of ROU is far from solved. Natural extension to the work presented in this paper may include: (1) inventing new summary data structures and indexing techniques to handle negative keyword conditions; (2) introducing ranking schemes to identify results that are most interesting and develop approximation semantics and algorithms for identifying top-k results; (3) exploring parallel implementations for scalability; and (4) extending query semantics for other types of interesting keyword queries.

7. REFERENCES

[1] C. Bron and J. Kerbosch. Finding all cliques of an undirected graph (algorithm 457). *Commun. ACM*, 16(9):575–576, 1973.

[2] S. Cohen, J. Mamou, Y. Kanza, and Y. Sagiv. XSEarch: A Semantic Search Engine for XML. In *VLDB*, pages 45–56, 2003.

[3] B. Ding, J. X. Yu, S. Wang, L. Qin, X. Zhang, and X. Lin. Finding top-k min-cost connected trees in databases. In *ICDE*, pages 836–845, 2007.

[4] K. Golenberg, B. Kimelfeld, and Y. Sagiv. Keyword proximity search in complex data graphs. In *SIGMOD*, pages 927–940, 2008.

[5] M. Kargar and A. An. Keyword search in graphs: Finding r-cliques. *PVLDB*, 4(10):681–692, 2011.

[6] G. Li, B. C. Ooi, J. Feng, J. Wang, and L. Zhou. EASE: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. In *SIGMOD*, pages 903–914, 2008.

[7] F. Liu, C. T. Yu, W. Meng, and A. Chowdhury. Effective keyword search in relational databases. In *SIGMOD*, pages 563–574, 2006.

[8] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to information retrieval*. Cambridge University Press, 2008.

[9] L. Qin, J. X. Yu, L. Chang, and Y. Tao. Querying communities in relational databases. In *ICDE*, pages 724–735, 2009.

[10] E. Tomita, A. Tanaka, and H. Takahashi. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theor. Comput. Sci.*, 363(1):28–42, 2006.

[11] M. J. Zaki, M. Peters, I. Assent, and T. Seidl. Clicks: An effective algorithm for mining subspace clusters in categorical datasets. *Data Knowl. Eng.*, 60(1):51–70, 2007.